Exact natural gradient in deep linear networks and application to the nonlinear case

Alberto Bernacchia
Department of Engineering
University of Cambridge
Cambridge, UK, CB2 1PZ
ab2347@cam.ac.uk

Máté Lengyel
Department of Engineering, University of Cambridge, Cambridge CB2 1PZ, UK
Department of Cognitive Science, Central European University, Budapest H-1051, Hungary
m.lengyel@eng.cam.ac.uk

Guillaume Hennequin
Department of Engineering
University of Cambridge
Cambridge, UK, CB2 1PZ
g.hennequin@eng.cam.ac.uk

Abstract

Stochastic gradient descent (SGD) remains the method of choice for deep learning, despite the limitations arising for ill-behaved objective functions. In cases where it could be estimated, the *natural* gradient has proven very effective at mitigating the catastrophic effects of pathological curvature in the objective function, but little is known theoretically about its convergence properties, and it has yet to find a practical implementation that would scale to very deep and large networks. Here, we derive an exact expression for the natural gradient in deep linear networks, which exhibit pathological curvature similar to the nonlinear case. We provide for the first time an analytical solution for its convergence rate, showing that the loss decreases exponentially to the global minimum in parameter space. Our expression for the natural gradient is surprisingly simple, computationally tractable, and explains why some approximations proposed previously work well in practice. This opens new avenues for approximating the natural gradient in the nonlinear case, and we show in preliminary experiments that our online natural gradient descent outperforms SGD on MNIST autoencoding while sharing its computational simplicity.

1 Introduction

Stochastic gradient descent (SGD) is used ubiquitously to train deep neural networks, due to its low computational cost and ease of implementation. However, long narrow valleys, saddle points and plateaus in the objective function dramatically slow down learning and often give the illusory impression of having reached a local minimum [Martens, 2010; Dauphin et al., 2014]. The natural gradient is an appealing alternative to the standard gradient: it accelerates convergence by using curvature information, it represents the steepest descent direction in the space of distributions, and is invariant to reparametrization of the network [Amari, 1998; Le Roux et al., 2008]. However, besides some numerical evidence, the exact convergence rate of natural gradient remains unknown, and its implementation remains prohibitive due to its very expensive numerical computation [Pascanu and Bengio, 2013; Martens, 2014; Ollivier, 2015].

In order to gain theoretical insight into the convergence rate of natural gradient descent, we analyze a deep (multilayer) linear network. While deep linear networks have obviously no practical relevance (they can only perform linear regression and are grossly over-parameterized, see below), their

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

optimization is non-convex and is plagued with similar pathological curvature effects as their nonlinear counterparts. Critically, the dynamics of learning in linear networks are exactly solvable, making them an ideal case study to understand the essence of the deep learning problem and find efficient solutions [Saxe et al., 2013]. Here, we derive an exact expression for the natural gradient in deep linear networks, from which we garner two major insights. First, we prove that the exact natural gradient leads to exponentially fast convergence towards the minimum achievable loss. This, to our knowledge, is the first case where a functional form for the natural gradient's convergence rate has been obtained for an arbitrarily deep multilayer network, and it confirms the long-standing conjecture that the natural gradient mitigates the problem of pathological curvature [Pascanu and Bengio, 2013; Martens, 2014] (and indeed, annihilates it completely in the linear case). Second, our exact solution reveals that the natural gradient can be computed much more efficiently than previously thought. By definition, the natural gradient is the product of the inverse of the $P \times P$ Fisher information matrix F with the P-dimensional gradient vector, where P is the number of network parameters (often in the millions) [Yang and Amari, 1998; Amari et al., 2000; Park et al., 2000]. In contrast, our expression exploits the structure of degeneracies in F and requires computing a similar matrix-vector product but in dimension N, the *number of neurons in each layer* (in the tens/hundreds). Although this simple expression does not formally apply to the nonlinear case, we adapt it to nonlinear deep networks and show that it outperforms SGD on the MNIST autoencoder problem.

Our exact expression for the natural gradient suggests retrospective theoretical justifications for several previously proposed modifications of standard gradient descent that empirically improved its convergence. In particular, we revisit previous approximations of the Fisher matrix (in the nonlinear case) based on block-diagonal truncations, and provide a possible explanation for their performance (K-FAC, [Martens and Grosse, 2015; Grosse and Martens, 2016; Ba et al., 2016], see also [Heskes, 2000; Povey et al., 2014; Desjardins et al., 2015]). We show that, even in the simple linear case, the exact inverse Fisher matrix is not block-diagonal and the contributions of the off-diagonal blocks to the natural gradient have the same order of magnitude as the on-diagonal blocks. Therefore, contrary to what has been proposed previously, the off-diagonal blocks cannot in principle be neglected. Instead, our analysis reveals that, when taking the inverse and multiplying by the gradient, the off-diagonal blocks of F contribute the exact same terms as the diagonal blocks. This observation is at the core of the surprisingly efficient yet exact way of computing the natural gradient that we propose here.

Finally, our algebraic expression for the natural gradient exhibits similarities with recent, biologically-inspired backpropagation algorithms. To obtain the natural gradient, we show that the error must back-propagate through the (pseudo-)inverses of the weight matrices, rather than their transposes. Multiplication by the matrix pseudo-inverse emerges automatically in algorithms where both forward and backward weights are free parameters [Lillicrap et al., 2016; Luo et al., 2017].

2 Natural gradient in deep networks

We consider the problem of learning an input-output relationship on the basis of observed data samples $\{(x_i, y_i)\}$ (input-output pairs) drawn from an underlying, unknown distribution $p^\star(x, y)$. This is achieved by a deep discriminative model, which, given an input x, specifies a conditional density $q_\theta(y|x)$ over possible outputs y, parameterized by the output layer of a deep network with a set of parameters θ. Specifically, the input vector $x \in \mathbb{R}^{n_0}$ propagates through a network of L layers according to:

$$x_i = \phi_i \left(W_i \, x_{i-1} + b_i \right) \qquad i = 1, \ldots, L \tag{1}$$

where $x_i \in \mathbb{R}^{n_i}$ is the output of layer i (which then serves as an input to layer $i+1$), $W_i \in \mathbb{R}^{n_i \times n_{i-1}}$ is a weight matrix into layer i, $b_i \in \mathbb{R}^{n_i}$ is a vector of bias parameters, ϕ_i is a function applied element-wise to its vector argument, and x_0 is defined as equal to the input x. The set of parameters θ includes all the elements of the weight matrices and bias vectors of all layers, for a total of P parameters. For ease of notation, in the following we include the bias vector b_i for each layer as an additional row in W_i, and augment the activation vector x_{i-1} accordingly with one constant component equal to one. The output of the last layer is x_L: it depends on all parameters θ and determines the conditional density $q_\theta(y|x)$, which we assume here is a Gaussian with a mean determined by x_L and a constant covariance matrix, $\tilde{\Sigma}$. Our theoretical results will be obtained for linear networks ($\phi_i(x) = x$ and $b_i = 0$), but we later return to nonlinear networks in numerical simulations.

The above model specifies a joint distribution of input/output pairs, i.e. $p_\theta(x, y) = q_\theta(y|x)\, q^\star(x)$ where $q^\star(x) = \int dy\, p^\star(x, y)$ is the marginal distribution of the input and does not depend on the parameters θ. The network is trained via maximum likelihood, i.e. by minimizing the following loss function:

$$\mathcal{L}(\theta) = \langle -\log p_\theta(x, y)\rangle_{p^\star} = \langle -\log q_\theta(y|x)\rangle_{p^\star} + \text{const} \tag{2}$$

where the average is over the true distribution $p^\star(x, y)$. In the following, we will use the shorthand notation $\ell(\theta|x, y) = \log p_\theta(x, y)$. Note that, in this setting, maximum likelihood is equivalent to minimizing the KL divergence $D_{\text{KL}}(p^\star \| p_\theta)$ between the true distribution $p^\star(x, y)$ and the model distribution $p_\theta(x, y)$.

A common way of minimizing $\mathcal{L}(\theta)$ is gradient descent, i.e. parameter updates of the form:

$$\frac{d\theta}{dt} \propto -\frac{\partial \mathcal{L}}{\partial \theta} = \left\langle \frac{\partial \ell(\theta|x, y)}{\partial \theta} \right\rangle_{p^\star} \tag{3}$$

where t denotes time elapsed in the optimization process. Although the theory of natural gradient we develop below applies to this continuous-time formulation [Mandt et al., 2017], numerical experiments are performed by discretizing Eq. 3 and setting a finite learning rate parameter. The dynamics of Eq. 3 are guaranteed to decrease the loss function in continuous time when the expectation over $p^\star$ can be evaluated exactly; in practice, these dynamics are approximated by sampling from $p^\star$ using a batch of training data points, and using a small but finite time step (learning rate) – this is SGD.

The natural gradient corresponds to a modification of Eq. 3, which consists of multiplying the (negative) gradient by the inverse of the Fisher information matrix F:

$$\frac{d\theta}{dt} \propto -F(\theta)^{-1} \frac{\partial \mathcal{L}}{\partial \theta} \tag{4}$$

where the Fisher information matrix $F \in \mathbb{R}^{P \times P}$ is defined as

$$F(\theta) = \left\langle \frac{\partial \ell}{\partial \theta} \cdot \frac{\partial \ell}{\partial \theta}^T \right\rangle_{p_\theta} \tag{5}$$

Note that the average is taken over the model distribution $p_\theta(x, y) = q_\theta(y|x) q^\star(x)$, rather than the true distribution $p^\star(x, y)$. Since the Fisher matrix is positive definite, the natural gradient also guarantees decreasing loss in continuous time. The Fisher information matrix quantifies the accuracy with which a set of parameters can be estimated by the observation of data, and the natural gradient thus rescales the standard gradient accordingly. The natural gradient has a number of desirable properties: it corresponds to steepest gradient descent in the space of distributions $p_\theta(x, y)$, it is parameterization-invariant, and affords good generalization performance [Amari, 1998; Le Roux et al., 2008]. Moreover, natural gradient descent can be regarded as a second-order method in the space of parameters (e.g. it reduces to the Gauss-Newton method in some cases [Pascanu and Bengio, 2013; Martens, 2014]).

3 Exact natural gradient for deep linear networks and quadratic loss

In this paper, we focus on regression problems where the conditional model distribution $q_\theta(y|x)$ is Gaussian, with a mean equal to the output x_L of the last layer of a deep network and some covariance $\tilde{\Sigma}$. Note that other types of distributions can also be used, e.g. a categorical distribution parameterized by the output of a final softmax layer to address classification problems. Using Eq. 2, the loss function for a Gaussian distribution is equal to the mean squared error weighted by the inverse covariance

$$\mathcal{L} = \frac{1}{2} \left\langle (y - x_L)^T \, \tilde{\Sigma}^{-1} \, (y - x_L) \right\rangle_{p^\star} + \text{const} \tag{6}$$

where the loss depends on the parameters of the deep network through the conditional mean x_L, and the constant includes all the terms that do not depend on x_L and thus on the network parameters. Using the expression for the loss, we can compute the gradient with respect to the weight matrix into layer i, as

$$\frac{\partial \mathcal{L}}{\partial W_i} = -\left\langle e_i \, x_{i-1}^T \right\rangle_{p^\star} \tag{7}$$

where $e_i \in \mathbb{R}^{n_i}$ is the error propagated backward to layer i (see below, Eq. 8), and x_{i-1} is the activation of layer $i-1$ propagated forward (Eq. 1). Note that this expression for the gradient is a matrix of the size of W_i. The expression for the backpropagated error is given by

$$\begin{aligned} e_L &= \phi'_L \circ \left[\tilde{\Sigma}^{-1}\left(y - x_L\right)\right] \\ e_i &= \phi'_i \circ \left[W_{i+1}^T\, e_{i+1}\right] \qquad i = 1, \ldots, L-1 \end{aligned} \tag{8}$$

where the symbol $\circ$ denotes the element-wise (Hadamard) vector product, and ϕ'_i denotes the scalar derivative of ϕ_i, evaluated at its argument defined in Eq. 1. The gradient is computationally cheap to evaluate, since a single forward pass is used to compute the activations x_i of all layers, and a single backward pass is used to compute the corresponding errors e_i.

It is currently unknown if the natural gradient affords an expression as simple and computationally cheap as those used to evaluate the standard gradient (Eqs. 7-8). Here, we derive such an expression in the case of a deep linear network. We thus take $\phi_i(x) = x$ ($\forall i$), and set the bias vectors to zero without loss of generality if the input has zero mean, $\langle x \rangle_{q^\star} = 0$. Using Eq. 1, the activation of the last layer is therefore equal to

$$x_L = \left(W_L \cdot W_{L-1} \cdots W_2 \cdot W_1\right) x = W x \tag{9}$$

where we defined the total weight matrix product W in the last expression, equal to the chain of matrix multiplications along all layers $1, 2, \ldots, L$. This expression makes obvious the uselessness of having multiple, successive linear layers, as their combined effect reduces to a single one. However, the dynamics of learning (e.g. by gradient descent) in each layer is highly nonlinear, while being amenable to analytical solutions [Saxe et al., 2013].

In the Supplementary Material, we calculate the Fisher information matrix F for a deep linear network. As expected, the Fisher matrix is singular, due to the aforementioned parameter redundancies, and therefore the model cannot be identified in certain directions in parameter space. In particular, the total number of parameters is $P = \sum_{i=1}^{L} n_i\, n_{i-1}$, where $n_i \times n_{i-1}$ are the dimensions of matrix W_i in layer i, and the total number of parameters is obtained by summing over all layers. However, there are only $n_L\, n_0$ independent parameters, which are the dimensions of the total product of weight matrices, W, in Eq. 9. Thus, the Fisher matrix is of rank $n_L\, n_0$ at most, and is therefore necessarily singular.

Due to the above singularity, the matrix inversion prescribed by Eq. 4 to obtain the natural gradient must be replaced by a generalized inverse (indeed, this is the appropriate way of dealing with this singularity, and it comes from the interpretation of the Fisher matrix as a metric in the space of distributions [Pascanu and Bengio, 2013]). Note that there exist an infinite number of generalized inverses. Our main result is proving that under the natural gradient, the dynamics of p_θ, and therefore also the dynamics of the loss function, are identical for all possible generalized inverses of the Fisher matrix (Supplementary Material). Moreover, any choice thereof leads to exponentially fast convergence towards the minimum loss. Critically though, all those possible generalized inverses might differ greatly in the simplicity and associated computational cost of the resulting parameter updates. We find that one particular generalized inverse leads to the following, remarkably simple expression:

$$\frac{dW_i}{dt} \propto \frac{1}{L} \left\langle e_i\, e_i^T \right\rangle_{p_\theta}^{-1} \left\langle e_i\, x_{i-1}^T \right\rangle_{p^\star} \left\langle x_{i-1}\, x_{i-1}^T \right\rangle_{p_\theta}^{-1} \tag{10}$$

This expression is equal to the standard gradient (middle term, cf. Eq. 7), multiplied by the inverse covariance of both the backward error e_i (left) and forward activation x_{i-1} (right). Note that these covariances correspond to averages over the model distribution $p_\theta(x, y)$, and *not* the true distribution $p^\star(x, y)$. When the inverses of those covariances do not exist, it is their Moore-Penrose pseudoinverse that must be used instead (Supplementary Material). As expected for the natural gradient, Eq. 10 is dimensionally consistent (weight updates have the same "units" as the weight matrices themselves), and is covariant for linear transformations.

At first glance, Eq. 10 requires two matrix multiplications and inversions per layer, which make it more costly than standard gradient descent. However, if the expectation over $p^\star$ is approximated by sampling as in SGD, then one only needs to perform two matrix-vector products, and make rank-1 updates of W_i, which brings the computational cost down to that of SGD. Finally, one can either (pseudo-)invert the two covariance matrices in Eq. 10 e.g. using an SVD (scales poorly with layer size, but otherwise cache efficient), or directly estimate their inverses using Sherman-Morrison updates (in which case the complexity scales with both layer size and network depth in the same way as for SGD). We discuss these practical issues further below.

4 Analytic expression for convergence rate

In this section, we provide a simplified derivation of the exponential decrease of the loss function under the the natural gradient updates given by Eq. 10, which are based on a particular form of the generalized inverse of F. The equation for the natural gradient is given by Eq. 16 below, which corresponds to Eq. 57 in the Supplementary Material. A more general derivation of the exponential decrease of the loss function is given in the Supplementary Material, where we show that the same exponential decay of the loss holds for all possible generalized inverses.

Using Eqs. 1 and 8, the forward activation and backward error in a linear network are given by

$$x_{i-1} = (W_{i-1} \cdots W_1)\, x \tag{11}$$

$$e_i = (W_L \cdots W_{i+1})^T\, \tilde{\Sigma}^{-1}\, (y - x_L) \tag{12}$$

Using Eq. 7, the gradient of the loss function is equal to the averaged outer product of the backward error and the forward activity, namely

$$\frac{\partial \mathcal{L}}{\partial W_i} = -\left\langle e_i\, x_{i-1}^T \right\rangle_{p^\star} = -\,(W_L \cdots W_{i+1})^T\, \tilde{\Sigma}^{-1} \left\langle (y - x_L)\, x^T \right\rangle_{p^\star} (W_{i-1} \cdots W_1)^T \tag{13}$$

In order to derive the natural gradient update, we calculate the covariance matrices in Eq. 10. The covariance of the backward error is equal to

$$\begin{aligned} \left\langle e_i e_i^T \right\rangle_{p_\theta} &= (W_L \cdots W_{i+1})^T\, \tilde{\Sigma}^{-1} \left\langle (y - x_L)\, (y - x_L)^T \right\rangle_{p_\theta} \tilde{\Sigma}^{-1}\, (W_L \cdots W_{i+1}) \\ &= (W_L \cdots W_{i+1})^T\, \tilde{\Sigma}^{-1}\, (W_L \cdots W_{i+1}) \end{aligned} \tag{14}$$

The second line results from averaging over the model distribution $p_\theta(x, y) = q_\theta(y|x) q^\star(x)$: the first average over the conditional distribution $q_\theta(y|x) = \mathcal{N}(y; x_L, \tilde{\Sigma})$ yields the covariance $\tilde{\Sigma}$ itself, and the latter does not depend on the input (making the average over $q^\star(x)$ unnecessary). Similar arguments lead to the covariance of the forward activity:

$$\left\langle x_{i-1} x_{i-1}^T \right\rangle_{p_\theta} = (W_{i-1} \cdots W_1) \left\langle x x^T \right\rangle_{p_\theta} (W_{i-1} \cdots W_1)^T = (W_{i-1} \cdots W_1)\, \Sigma\, (W_{i-1} \cdots W_1)^T \tag{15}$$

where $\Sigma = \left\langle x x^T \right\rangle_{q^\star}$ is the covariance of the input (the average is taken over the model distribution p_θ, but $x x^T$ depends on the input distribution $q^\star$ only).

In order to compute the natural gradient of Eq. 10, we need to invert the covariances in Eqs. 14 and 15. However, they may not be invertible, except in special cases, such as when all weight matrices are square and invertible, and when both Σ and $\tilde{\Sigma}$ are full rank. We consider this simple case first, and then address the general case of non-square matrices. If we can invert explicitly the relevant covariance matrices, substituting into Eq. 10, along with Eq. 13, yields updates of the form

$$\frac{dW_i}{dt} \propto \frac{1}{L}\, (W_L \cdots W_{i+1})^{-1} \left\langle (y - x_L)\, x_0^T \right\rangle_{p^\star} \Sigma^{-1}\, (W_{i-1} \cdots W_1)^{-1} \tag{16}$$

This equation does not immediately suggest any advantage with respect to standard gradient descent. However, it is revealing to derive the dynamics of the total weight matrix product, $W = W_L \cdots W_1$, which represents the net input-output mapping performed by the network. Using the product rule of differentiation:

$$\frac{dW}{dt} = \sum_{i=1}^{L} (W_L \cdots W_{i+1})\, \frac{dW_i}{dt}\, (W_{i-1} \cdots W_1) \tag{17}$$

Substituting the expression for the update, Eq. 16, and using $x_L = W x_0$ we obtain

$$\frac{dW}{dt} \propto -W + \left\langle y x^T \right\rangle_{p^\star} \Sigma^{-1} \tag{18}$$

Thus, under natural gradient descent in continuous time, the total weight matrix obeys first order dynamics, and therefore converges exponentially fast towards $\left\langle yx^T \right\rangle \Sigma^{-1}$, which is indeed the least squares solution to the linear regression problem [Bishop, 2016]. Since the loss is a quadratic function of W (cf. Eq. 6), Eq. 18 also proves that the loss decays exponentially towards zero under natural gradient descent. This result holds provided that the network parameters are not initialized at a saddle point (for example, weights should not be initialized at zero).

When the covariances in Eqs. 14 and 15 cannot be inverted, e.g. when the weight matrices are not square (the network is contracting, expanding, or contains a bottleneck), we show in the Supplementary Material (Eq. 68) that the Moore-Penrose pseudo-inverse must be used instead, inducing similar dynamics for W:

$$\frac{dW}{dt} \propto -W + \frac{1}{L} \sum_{i=1}^{L} P_i^a \left\langle yx^T \right\rangle_{p^\star} \Sigma^{-1} P_i^b \tag{19}$$

Here, P_i^a and P_i^b are projection matrices that express the way in which the network architecture constrains the space of solutions that the network is allowed to reach. For example, if the network has a bottleneck, the total matrix W will only be able to attain a low-rank approximation of the optimal solution to the regression problem, $\left\langle yx_0^T \right\rangle \Sigma^{-1}$. Note, for example, that $P_i^a = I$ (identity matrix) for a non-expanding network, while $P_i^b = I$ for a non-contracting network.

5 Implementation of natural gradient descent and experiments

Similar to SGD, we approximate the average over $p^\star$ in Eq. 7 by using mini-batches of size M. For each input mini-batch x, we use the forward activations (already calculated in the forward pass to get the gradient information) to estimate $\Lambda_i = \left\langle x_{i-1} x_{i-1}^T \right\rangle_{p_\theta}$. Then, for the same input mini-batch, we also sample K times from the model predictive distribution $q_\theta(y|x)$, use these outputs as targets, and perform the corresponding K backward passes to obtain KM backpropagated error samples used to estimate $\tilde{\Lambda}_i = \left\langle e_i e_i^T \right\rangle_{p_\theta}$. Note that the true outputs of the training set are only used to compute (a stochastic estimate of) the gradient of the loss function, but never used to estimate Λ_i nor $\tilde{\Lambda}_i$ (indeed, these are averages over p_θ, not $p^\star$). In practice, we find that $K = 1$ suffices.

Weights are updated according to Eq. 10, discretized using a small time step (learning rate α). Inspired by the interpretation of NGD as a second-order method [Martens, 2014], we also incorporate a Levenberg-Marquardt-type damping scheme: at each iteration k, we add $\sqrt{\lambda_k} I$ to both covariance matrices Λ_i and $\tilde{\Lambda}_i$ prior to inverting them, where λ_k is an adaptive damping factor. Note that this is not equivalent to adding λ_k to the Fisher matrix. Nevertheless, it does become equivalent to a small SGD step in the limit of large damping factor λ_k. Therefore, at iteration k we update the synaptic weights in layer i according to

$$\Delta W_i = \frac{\alpha}{L} \left(\tilde{\Lambda}_i + \sqrt{\lambda_k} I \right)^{-1} \left\langle e_i x_{i-1}^T \right\rangle_{p^\star} \left(\Lambda_i + \sqrt{\lambda_k} I \right)^{-1} \tag{20}$$

We update λ_k in each iteration to reflect the ratio ρ_k between i) the actual decrease in the loss resulting from the latest damped NG parameter update, and ii) the decrease predicted by a quadratic approximation to the loss[1]. The damping factor is updated as follows:

$$\lambda_{k+1} = \begin{cases} \frac{3\lambda_k}{2} & \text{if } \rho_k < 0.25 \\ \frac{2\lambda_k}{3} & \text{if } \rho_k > 0.75 \end{cases} \tag{21}$$

We experimented with deep networks (linear and nonlinear) trained on regression problems (Fig. 1). First, we trained three linear networks to recover the mappings defined by random networks in their model class. The first network (Fig. 1A) had $L = 16$ layers of the same size $n_i = 20$. The second (Fig. 1B) had $L = 16$ layers, of size $20(\text{input}), 30, 40, \ldots, 100, \ldots, 30, 20$. While these two networks

[1]Here, the quadratic approximation is implicitly defined as the quadratic function whose minimization by the Newton method would require a step in the direction of ΔW_i, the momentary update taken by our damped NGD step. The predicted decrease in loss under such a quadratic approximation is cheap to compute: if ΔW_i is the NG update for layer i, then the predicted decrease in the loss is given by $\left(-\alpha + \frac{\alpha^2}{2}\right) \sum_i \operatorname{tr} \left(\Delta W_i^T \left\langle e_i x_{i-1}^T \right\rangle_{p^\star} \right)$.

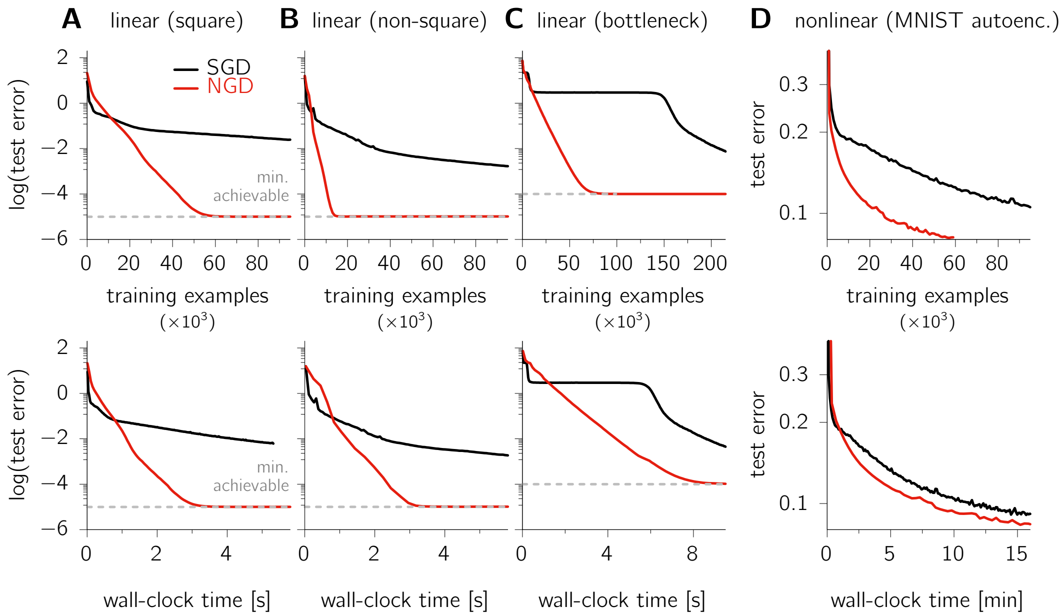

Figure 1: **Natural gradient in deep networks**. **(A–C)** Dynamics of the loss function under optimization with SGD (black) and NGD (red), for three deep linear networks with different architectures (see main text for details). Training time is reported both as number of training examples seen so far (top) and wall-clock time (bottom). Both optimization algorithms start from the same initial network parameters. Dashed gray lines denote the smallest possible loss, determined by the variance of the true underlying conditional density of $y|x$. **(D)** Test error for MNIST autoencoding in a deep nonlinear network (see main text for details); colors are the same as in (A–C). SGD parameters: $M = 20$, learning rate α optimized by grid search (A and B: $\alpha = 0.08$; C: $\alpha = 0.02$; D: $\alpha = 0.04$). NGD parameters: $\alpha = 1$, $M = 1000$.

were over-parameterized, our third network (Fig. 1C) was an under-parameterized bottleneck with steep fan-in and fan-out, with $L = 12$ layers of size $200(\text{input}), 80, 34, 20, 10, 5, 2, 5, \ldots, 80, 200$. For each architecture, we generated a network with random parameters $\theta^\star$ and used it as the reference mapping to be learned. We generated a training set of 10^4 examples, and a test set of 10^3 examples, by propagating inputs drawn from a correlated Gaussian distribution $q^\star(x) = \mathcal{N}(x; 0, \Sigma)$ through the network, and sampling outputs from a Gaussian conditional distribution $q_{\theta^\star}(y|x)$ with covariance $\tilde{\Sigma} = 10^{-6} I$. We generated Σ to have random (orthogonal) eigenvectors and eigenvalues that decayed exponentially as e^{-5i/n_0}.

We compared SGD (with minibatch size $M = 20$, and learning rate optimized via grid search) and online natural gradient (with minibatch size $M = 1000$). For both tasks, SGD made fast initial progress, but slowed down dramatically very soon. In contrast, as predicted by our theory, natural gradient descent caused the test error to decrease exponentially and reach the minimum achievable loss (limited by $\tilde{\Sigma}$) after only a few passes through the training set (Fig. 1A-C, top).

As a preliminary extension to the nonlinear case, we also trained a nonlinear network with eight layers of size $784(\text{input}), 400, 200, 100, 50, 100, \ldots, 784$, to perform autoencoding of the MNIST dataset (Fig. 1D). All layers had $\phi_i(x) = \tanh(x)$, except for the final linear layer. We compared standard SGD (with $M = 20$ and α optimized by grid search) to our proposed natural gradient method (Eq. 10, with adaptive damping and no further modification). We set $\alpha = 1$, $M = 1000$ and $K = 1$. Despite our NGD steps only approximating the true natural gradient, it outperformed SGD in terms of data efficiency (Fig. 1D, top). Owing to the size of the input layer, our implementation of NGD via direct inversion of the relevant covariance matrices outperformed SGD only modestly in wall-clock time (Fig. 1D, bottom). We discuss alternative implementations below.

6 Related work

Diagonal approximations As reviewed in Martens [2014], some recent popular methods can be interpreted as diagonal approximations to the Fisher matrix F, such as AdaGrad [Duchi et al., 2011], AdaDelta [Zeiler, 2012], and Adam [Kingma and Ba, 2014]. Those methods are computationally cheap, but do not capture pairwise dependencies between parameters. In theory, faster learning could be obtained by leveraging full curvature information, which requires moving away from a purely diagonal approximation of F. However, this is computationally intensive for at least two reasons: i) the Fisher matrix is large, often impossible to store, let alone to invert, and ii) even if one could compute F^{-1}, the natural gradient would still require $\mathcal{O}(P^2)$ operations (where P is the number of parameters). Much of the recent literature has focused on ways of mitigating this complexity. For example, in cases where it can be stored, F^{-1} can be estimated directly using the Sherman-Morrison lemma [Amari et al., 2000]. When it cannot be stored, one can approximate the natural gradient directly via conjugate gradients, exploiting fast methods for computing Fv products (as in Hessian-free and Gauss-Newton optimization [Martens, 2010; Pascanu and Bengio, 2013; Martens and Grosse, 2015; Vinyals and Povey, 2012]). Often, however, many steps of conjugate gradients must be performed at each training iteration to make good progress on the loss. Here, we have obtained the surprising result that $F^{-1}v$ products can in fact be obtained directly (in linear networks), at almost the same cost as Fv.

Block-diagonal approximations In order to obtain an expression for the natural gradient that would be computationally cheap and feasible for practical applications, previous studies suggested a block-diagonal approximation to the inverse Fisher information matrix, in the nonlinear case (K-FAC, [Martens and Grosse, 2015; Grosse and Martens, 2016; Ba et al., 2016], see also [Heskes, 2000; Povey et al., 2014; Desjardins et al., 2015]). In general, there is no formal justification for assuming that the Fisher information matrix (or its inverse) is block diagonal. In our deep linear network model, we show in the Supplementary Material (cf. Eq. 42) that the (i, j)-block of the exact Fisher information matrix (corresponding to the weight matrices of layers i and j), is equal to

$$F_{ij} = \left\langle x_{i-1} x_{j-1}^T \right\rangle_{p_\theta} \otimes \left\langle e_i e_j^T \right\rangle_{p_\theta} \tag{22}$$

There is no reason to expect that this expression is zero for $i \neq j$, unless the outputs x_i or the errors e_i are uncorrelated across all pairs of layers, and indeed Eq. 42 in the Supplementary Material shows that it is not zero. Nevertheless, if we choose to ignore this fact and set $F_{ij} = 0$ for $i \neq j$, then inverting the Fisher matrix (by inverting separately each diagonal block F_{ii}) generates an expression proportional to the exact natural gradient of Eq. 10.

In order to understand this puzzling observation, we recall that the exact Fisher is singular, and we chose a specific form for the generalized inverse F^g in order to derive Eq. 10 (while noting that the dynamics of the loss is the same for all possible inverses). In the Supplementary Material (cf. Eq. 59), we note that the (i, j)-block of this specific generalized inverse is equal to

$$(F^g)_{ij} = \frac{1}{L^2} \left\langle x_{j-1} x_{i-1}^T \right\rangle_{p_\theta}^{-1} \otimes \left\langle e_j e_i^T \right\rangle_{p_\theta}^{-1} \tag{23}$$

Thus each block of the inverse Fisher is equal to the inverse of the corresponding block of the (transposed) Fisher matrix (note that we assumed square and invertible blocks). However, the inverse Fisher is not block-diagonal either, thus it remains unclear why the approximation works. The solution to this puzzle is the following. In deriving the natural gradient update for layer i, we must multiply an entire row of blocks of the inverse Fisher by the gradient across all layers. Surprisingly, each of these blocks makes exactly the same contribution to the natural gradient (Eq. 60 in the Supplementary Material). Thus, we can get away with computing the single contribution of the diagonal block for each row, and simply multiply that by the number of blocks in the row. This is of course equivalent, though only fortuitously so, to making a block-diagonal approximation of F in the first place. Therefore, somewhat incidentally, a block-diagonal approximation is expected to perform just as well as the full matrix inversion.

Whitening and biological algorithms Our expression for the natural gradient offers post-hoc justifications for some recently proposed modifications of the standard gradient, whereby the forward activation and backward errors are whitened prior to being multiplied to obtain the gradient at each layer [Desjardins et al., 2015; Fujimoto and Ohira, 2018]. In our method, these vectors are also rescaled, albeit with their inverse covariances instead of the square root thereof (Eq. 10; see also Heskes [2000]; Martens and Grosse [2015]). Notably, this form of rescaling is equivalent to backpropating the error through the (pseudo)-inverses of the weight matrices, rather than their transpose (Eq. 16); interestingly, this strategy also tends to emerge in more biologically plausible algorithms in which both forward and backward weights are free parameters [Lillicrap et al., 2016; Luo et al., 2017].

7 Conclusions

We computed the natural gradient exactly for a deep linear network with quadratic loss function. We showed that the natural gradient is not unique in this case, because the Fisher information is singular due to over-parameterization. Surprisingly, we found that the loss function has the same convergence properties for all possible natural gradients, i.e. as obtained by any generalized inverse of the Fisher matrix. Indeed, one of our main results is the first exact solution for the convergence rate of the loss function under natural gradient descent, for a linear multilayer network: exponential decrease towards the minimum loss. This result backs up empirical claims of the natural gradient efficiently optimizing deep networks; in the deep linear case, we find that it solves the problem of pathological curvature entirely [Pascanu and Bengio, 2013; Martens, 2014]. Our results also consolidate deep linear networks as a useful case study for advancing the theory of deep learning. While Saxe et al. [2013] used linear theory to propose new ways of initializing neural networks, we have used it to propose a new, efficient optimization algorithm. We found that natural gradient updates afford an unexpectedly cheap form, with similar computational requirements as plain SGD.

Compared with the size of deep neural networks currently used, our application concerned relatively small networks of at most a few hundreds neurons per layer. Our current implementation based on direct inversion of Λ and $\tilde{\Lambda}$ in Eq. 20 may scale poorly (in wall-clock time) as the layer sizes increase. In this case, matrix pseudo-inversion in Eq. 10 could be performed using randomized SVD algorithms [Halko et al., 2011]. Alternatively, direct estimation of those matrix inverses via the Sherman-Morrison (SM) lemma should scale better [Amari et al., 2000], which we have confirmed in preliminary simulations. As SM updates tend to be less cache-efficient than direct inversion (they require many matrix-vector products instead of fewer matrix-matrix products), they may only benefit performance for very large layers. Moreover, more work is needed to incorporate adaptive damping into SM estimation of inverse covariances.

Our analytical results were derived for continuous time optimization dynamics. While we presented numerical evidence showing that a discrete-time implementation of NGD performs well, and it indeed shows the exponential decrease of the loss function predicted by our theory, further work is necessary in order to derive principled methods for discretizing the parameter updates [Martens, 2014].

Our core results relied exclusively on linear activation functions. While we have had some success in training nonlinear networks using Eq. 10 as a drop-in replacement for SGD (Fig. 1D), much remains to be done to make our algorithm effective in general deep learning settings. Improvements could be made to our adaptive damping scheme, for example through asymmetric damping of the covariance matrices Λ_i and $\tilde{\Lambda}_i$ in Eq. 20 as proposed by Martens and Grosse [2015]. More generally, deeper links need to be established between our linear NGD theory and systematic methods based on Kronecker factorizations (K-FAC [Martens and Grosse, 2015; Grosse and Martens, 2016; Ba et al., 2016]). A key insight from our analysis is that there exist infinitely many ways of computing the NG in linear deep networks (and probably also in nonlinear networks in which the Fisher matrix has been found to be near-degenerate [Le Roux et al., 2008]). While all of these different methods result in fast learning with identical dynamics for the loss function, their computational complexity may differ greatly. Moreover, there may be more than one computationally tractable method (such as the one we have used here), and in turn, some of these may be more suitable than others for use as a drop-in replacement to SGD in nonlinear networks. We suggest that further analysis of deep linear networks will prove invaluable for deriving efficient new training algorithms.

Acknowledgments

This work was supported by Wellcome Trust Seed Award 202111/Z/16/Z (G.H.) and Wellcome Trust Investigator Award 095621/Z/11/Z (A.B.,M.L.).

References

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276.

Amari, S.-I., Park, H., and Fukumizu, K. (2000). Adaptive method of realizing natural gradient learning for multilayer perceptrons. *Neural Computation*, 12(6):1399–1409.

Ba, J., Grosse, R., and Martens, J. (2016). Distributed second-order optimization using kronecker-factored approximations. In *Proc. 5rd Int. Conf. Learn. Representations*.

Bishop, C. M. (2016). *Pattern recognition and machine learning.* Springer-Verlag New York.

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941.

Desjardins, G., Simonyan, K., Pascanu, R., et al. (2015). Natural neural networks. In *Advances in Neural Information Processing Systems*, pages 2071–2079.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Fujimoto, Y. and Ohira, T. (2018). A neural network model with bidirectional whitening. In *International Conference on Artificial Intelligence and Soft Computing*, pages 47–57. Springer.

Grosse, R. and Martens, J. (2016). A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pages 573–582.

Halko, N., Martinsson, P.-G., and Tropp, J. A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288.

Heskes, T. (2000). On natural learning and pruning in multilayered perceptrons. *Neural Computation*, 12(4):881–901.

Kingma, D. P. and Ba, J. L. (2014). Adam: A method for stochastic optimization. In *Proc. 3rd Int. Conf. Learn. Representations*.

Le Roux, N., Manzagol, P.-A., and Bengio, Y. (2008). Topmoumoute online natural gradient algorithm. In *Advances in neural information processing systems*, pages 849–856.

Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. (2016). Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7:13276.

Luo, H., Fu, J., and Glass, J. (2017). Bidirectional backpropagation: Towards biologically plausible error signal transmission in neural networks. *arXiv preprint arXiv:1702.07097*.

Mandt, S., Hoffman, M. D., and Blei, D. M. (2017). Stochastic gradient descent as approximate bayesian inference. *The Journal of Machine Learning Research*, 18(1):4873–4907.

Martens, J. (2010). Deep learning via hessian-free optimization. In *International conference on machine learning*, volume 27, pages 735–742.

Martens, J. (2014). New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*.

Martens, J. and Grosse, R. (2015). Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417.

Ollivier, Y. (2015). Riemannian metrics for neural networks I: feedforward networks. *Information and Inference: A Journal of the IMA*, 4(2):108–153.

Park, H., Amari, S.-I., and Fukumizu, K. (2000). Adaptive natural gradient learning algorithms for various stochastic models. *Neural Networks*, 13(7):755–764.

Pascanu, R. and Bengio, Y. (2013). Revisiting natural gradient for deep networks. In *Proc. 2rd Int. Conf. Learn. Representations*.

Povey, D., Zhang, X., and Khudanpur, S. (2014). Parallel training of dnns with natural gradient and parameter averaging. In *Proc. 3rd Int. Conf. Learn. Representations*.

Saxe, A. M., McClelland, J. L., and Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *Proc. 2rd Int. Conf. Learn. Representations*.

Vinyals, O. and Povey, D. (2012). Krylov subspace descent for deep learning. In *Artificial Intelligence and Statistics*, pages 1261–1268.

Yang, H. H. and Amari, S.-i. (1998). Complexity issues in natural gradient descent method for training multilayer perceptrons. *Neural Computation*, 10(8):2137–2157.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

8 Supplementary Material: Calculation of the natural gradient

In this section we calculate exactly the natural gradient in a deep linear network with Gaussian noise. We show that the natural gradient is not unique, but the dynamics of the loss function does not depend on its choice: all possible natural gradients lead to exponentially fast minimization of the loss. We show that one specific choice of the natural gradient is equivalent to the rule studied in the main text, Eq. 10 and Eq. 16.

8.1 Standard gradient

Here we summarized the main assumptions of the model and calculate the gradient of the loss function. The output y of the deep network is specified by the distribution $q_\theta(y|x)$, conditioned on the input x, a Gaussian with mean x_L and covariance $\tilde{\Sigma}$, where x_L is the output of the last layer of the deep network, which depends on the parameters θ (all synaptic weights in this section). In a deep linear network, x_L is equal to

$$x_L = Wx \tag{24}$$

where W is the total weight matrix product across layers, given by

$$W = W_L \cdots W_1 \tag{25}$$

and L is the total number of layers. The number of neurons in layer i is n_i, and the number of components of the input is n_0; The matrix W_i has size $n_i \times n_{i-1}$. The log-likelihood of the conditional mean x_L, given the observed data (x, y), is equal to

$$\ell(x_L|x,y) = \log q_\theta\left(y|x\right) + \text{const} = -\frac{1}{2}\left(y - x_L\right)^T \tilde{\Sigma}^{-1}\left(y - x_L\right) + \text{const} \tag{26}$$

Note that the log-likelihood depends on the parameters W through the conditional mean x_L. Constant terms do not depend on x_L and therefore on W. The loss function is defined as minus the log-likelihood (only the relevant term) averaged over the true distribution $p^\star(x, y)$, namely

$$\mathcal{L}(W) = \left\langle -\ell(x_L|x,y)\right\rangle_{p^\star} = \frac{1}{2}\left\langle \left(y - Wx\right)^T \tilde{\Sigma}^{-1}\left(y - Wx\right)\right\rangle_{p^\star} \tag{27}$$

The goal is to minimize this loss function with respect to the parameters $W_1, \ldots, W_L$. For convenience of notation, we define the product of weight matrices *ahead* of a given layer i, equal to

$$W_i^a = W_L \cdots W_{i+1} \tag{28}$$

and the product of weight matrices *behind* a given layer i, equal to

$$W_i^b = W_{i-1} \cdots W_1 \tag{29}$$

Such that the total weight matrix product is rewritten as $W = W_i^a W_i W_i^b$ for any layer $i = 1, \ldots, L$. Using all definitions above, we calculate the gradient of the log likelihood with respect to weight matrix W_i in layer i, which is equal to

$$\frac{\partial \ell}{\partial W_i} = e_i x_{i-1}^T = W_i^{aT}\frac{\partial \ell}{\partial x_L} x^T W_i^{bT} \tag{30}$$

where the backward error and forward activity are equal to

$$e_i = W_i^{aT}\frac{\partial \ell}{\partial x_L} \qquad\qquad x_{i-1} = W_i^b x \tag{31}$$

respectively a vector of n_i and n_{i-1} components, and the gradient is a matrix of size $n_i \times n_{i-1}$. The gradient of the log likelihood with respect to the network output, for a Gaussian distribution, is equal to the error weighted by the (inverse) conditional covariance, namely

$$\frac{\partial \ell}{\partial x_L} = \tilde{\Sigma}^{-1}\left(y - x_L\right) \tag{32}$$

Standard gradient descent corresponds to updating the weights according to minus the gradient of the loss function, or plus the gradient of the averaged log-likelihood, namely

$$\frac{dW_i}{dt} \propto -\frac{\partial \mathcal{L}}{\partial W_i} = \left\langle \frac{\partial \ell}{\partial W_i}\right\rangle_{p^\star} = \left\langle e_i x_{i-1}^T\right\rangle_{p^\star} \tag{33}$$

8.2 Exact Fisher information matrix

The natural gradient is obtained by calculating and inverting the Fisher information matrix, and then multiplying the result by the gradient, as described by Eq. 4. Note that the gradient for layer i, Eq. 30, is a matrix of size $n_i \times n_{i-1}$; In order to calculate the Fisher information matrix, we need to express the gradient in vectorized form, where the columns of the matrix are piled up in a single column vector of $n_i n_{i-1}$ components. Using Eq. 30, the vectorized form of the gradient for layer i is equal to

$$\frac{\partial \ell}{\partial \text{Vec}\,(W_i)} = \text{Vec}\left(W_i^{aT} \frac{\partial \ell}{\partial x_L} x^T W_i^{bT} \right) = \tag{34}$$

$$= \left(W_i^b \otimes W_i^{aT} \right) \text{Vec}\left(\frac{\partial \ell}{\partial x_L} x^T \right) = \left(W_i^b x \right) \otimes \left(W_i^{aT} \frac{\partial \ell}{\partial x_L} \right) \tag{35}$$

where we used the definition of the Kronecker product $\otimes$. The Fisher information matrix F is given by Eq. 5; The outer products of gradients is calculated by noting that the vector of all parameters θ piles up the elements of all weight matrices, and is written in vectorized form as $\theta = \text{Vec}\,(W_1, \ldots, W_L)$. Therefore, the Fisher information matrix is equal to

$$F = \left\langle \frac{\partial \ell}{\partial \text{Vec}\,(W_1, \ldots, W_L)} \cdot \frac{\partial \ell}{\partial \text{Vec}\,(W_1, \ldots, W_L)}^T \right\rangle_{p_\theta} \tag{36}$$

where all columns of all matrices are piled up in a single vector with $P = \sum_{i=1}^{L} n_i n_{i-1}$ components, which represents the total number of parameters, and the Fisher matrix has size $P \times P$. In order to simplify the notation, we consider different blocks of this matrix, each block corresponding to a different pair of weight matrices (a different pair of layers). The (i,j)-block of the Fisher matrix F is given by

$$F_{ij} = \left\langle \frac{\partial \ell}{\partial \text{Vec}\,(W_i)} \cdot \frac{\partial \ell}{\partial \text{Vec}\,(W_j)}^T \right\rangle_{p_\theta} \tag{37}$$

This block has size $n_i n_{i-1} \times n_j n_{j-1}$. Using Eq. 35 and the mixed product property of the Kronecker product, this is equal to

$$F_{ij} = \left\langle \left(W_i^b x x^T W_j^{bT} \right) \otimes \left(W_i^{aT} \frac{\partial \ell}{\partial x_L} \frac{\partial \ell}{\partial x_L}^T W_j^a \right) \right\rangle_{p_\theta} = \tag{38}$$

$$= \left(W_i^b \otimes W_i^{aT} \right) \left\langle x x^T \otimes \frac{\partial \ell}{\partial x_L} \frac{\partial \ell}{\partial x_L}^T \right\rangle_{p_\theta} \left(W_j^b \otimes W_j^{aT} \right)^T \tag{39}$$

Again, this expression is averaged over the model distribution $p_\theta(x, y) = q_\theta(y|x) q^\star(x)$. Since the conditional distribution $q_\theta(y|x)$ is assumed Gaussian, the two factors inside angular brackets can be averaged separately; The left factor xx^T does not depend on y, thus the right factor can be separately averaged over $q_\theta(y|x)$. Once averaged, the right factor gives the inverse conditional covariance (see Eq. 32), which does not depend on x for a Gaussian distribution, and the left factor can be then averaged separately over x. Specifically, we have that

$$\left\langle x x^T \otimes \frac{\partial \ell}{\partial x_L} \frac{\partial \ell}{\partial x_L}^T \right\rangle_{p_\theta} = \left\langle x x^T \otimes \left\langle \frac{\partial \ell}{\partial x_L} \frac{\partial \ell}{\partial x_L}^T \right\rangle_{q_\theta} \right\rangle_{q^\star} = \left\langle x x^T \right\rangle_{q^\star} \otimes \tilde{\Sigma}^{-1} = \Sigma \otimes \tilde{\Sigma}^{-1} \tag{40}$$

where we defined $\Sigma = \langle x x^T \rangle$ as the covariance of the input (assumed centered - zero mean), and we used the fact that q_θ is Gaussian. Therefore, the (i,j)-block of the Fisher matrix is equal to

$$F_{ij} = \left(W_i^b \otimes W_i^{aT} \right) \left(\Sigma \otimes \tilde{\Sigma}^{-1} \right) \left(W_j^b \otimes W_j^{aT} \right)^T \tag{41}$$

By the mixed product property of the Kronecker product, this can be rewritten as

$$F_{ij} = \left(W_i^b \Sigma W_j^{bT} \right) \otimes \left(W_i^{aT} \tilde{\Sigma}^{-1} W_j^a \right) = \left\langle x_{i-1} x_{j-1}^T \right\rangle_{p_\theta} \otimes \left\langle e_i e_j^T \right\rangle_{p_\theta} \tag{42}$$

where we used the definition of backward error and forward activity, Eq. 31. This expression is exact in the linear Gaussian case studied here, and was derived previously as an approximation of the nonlinear case Heskes [2000]; Povey et al. [2014]; Desjardins et al. [2015]; Martens and Grosse [2015]; Grosse and Martens [2016]; Ba et al. [2016]).

8.3 Generalized inverse Fisher matrix

In order to calculate the natural gradient, the full Fisher matrix needs to be inverted, including all blocks. For convenience of notation, we define the following matrix

$$A = \left(\left(W_1^b \otimes W_1^{aT} \right)^T, \ldots, \left(W_L^b \otimes W_L^{aT} \right)^T \right)^T \tag{43}$$

Note that this matrix has size $P \times n_0 n_L$, and it represents the Jacobian of the function $W = W_L \cdots W_1$. We also define the following matrix, which corresponds to the Fisher information for a neural network with a single layer

$$B = \left(\Sigma \otimes \tilde{\Sigma}^{-1} \right) \tag{44}$$

which is square, positive definite and invertible, by the assumption that input and output are full rank, and has size $n_0 n_L \times n_0 n_L$. Then, putting together all blocks of Eq. 41, the full $P \times P$ Fisher matrix is equal to

$$F = ABA^T \tag{45}$$

By looking at the size of matrices A and B, it is clear that the Fisher matrix has at most rank $n_0 n_L$ and is not invertible. This is not surprising: since the feedforward network is linear, only $n_0 n_L$ parameters are independent, representing the product of the input and output components. However, we consider the generalized inverse of the Fisher matrix, and we ignore all directions in parameters space for which no information can be obtained.

We assume that A has maximum rank (it has independent columns), then it has a left inverse A^l, defined by $A^l A = 1$ (identity matrix). This holds typically when all weight matrices are full rank and the network does not have any bottleneck. Then, the generalized inverse of the Fisher matrix is

$$F^g = A^{lT} B^{-1} A^l \tag{46}$$

Note that the left inverse of A is non-unique, therefore the generalized inverse of the Fisher matrix is non-unique. However, we show below that the loss decays exponentially towards its minimum, regardless of the choice of A^l.

8.4 All natural gradients imply exponential decay for the loss

We seek to update the weights according to the natural gradient Eq. 4, which is given by the generalized inverse, Eq. 46, multiplied by the gradient across all weight matrices. Note that the vector of all parameters θ, piling up the elements of all weight matrices, is written in vectorized form as $\theta = \mathrm{Vec}\left(W_1, \ldots, W_L\right)$, and the update takes a similar vectorized form. Therefore, the natural gradient updates can be written as

$$\mathrm{Vec}\left(\frac{dW_1}{dt}, \ldots, \frac{dW_L}{dt} \right) \propto A^{lT} B^{-1} A^l \left\langle \frac{\partial \ell}{\partial \mathrm{Vec}\left(W_1, \ldots, W_L\right)} \right\rangle_{p^\star} \tag{47}$$

Note that the average of the gradient is taken over the true distribution $p^\star(x, y)$. This expression can be simplified by noting that the vectorized gradient across all layers, using Eqs.(35,43), can be written as

$$\frac{\partial \ell}{\partial \mathrm{Vec}\left(W_1, \ldots, W_L\right)} = A \; \mathrm{Vec}\left(\frac{\partial \ell}{\partial x_L} x^T \right) \tag{48}$$

Therefore, using the left inverse $A^l A = 1$, the natural gradient update is equal to

$$\mathrm{Vec}\left(\frac{dW_1}{dt}, \ldots, \frac{dW_L}{dt} \right) \propto A^{lT} B^{-1} \mathrm{Vec} \left\langle \frac{\partial \ell}{\partial x_L} x^T \right\rangle_{p^\star} \tag{49}$$

Although we eliminated one of the two left inverses in Eq. 47, an explicit expression for the left inverse A^l is still necessary in order to find a simple formula for the natural gradient. We give such an expression below, but first we show that the total weight matrix product converges exponentially to the optimal solution, if and only if it is updated using the natural gradient, for any choice of the left inverse.

The update for the total weight matrix product is given by

$$\frac{dW}{dt} = \sum_i W_i^a \frac{dW_i}{dt} W_i^b \tag{50}$$

Using again the definition of A, Eq. 43, we rewrite this update in vectorized form

$$\text{Vec}\left(\frac{dW}{dt}\right) = A^T \text{Vec}\left(\frac{dW_1}{dt}, \ldots, \frac{dW_L}{dt}\right) \tag{51}$$

Substituting Eq. 49 into Eq. 51, and using again the left inverse, $A^T {A^l}^T = \left(A^l A\right)^T = 1$, we find

$$\text{Vec}\left(\frac{dW}{dt}\right) \propto B^{-1} \text{Vec}\left\langle \frac{\partial \ell}{\partial x_L} x^T \right\rangle_{p^\star} \tag{52}$$

Perhaps surprisingly, the update for the total weight matrix product does not depend on the left inverse A^l, and thus it does not depend on the specific choice of the generalized inverse Fisher. Using Eqs.(32,44), the update of the total weight matrix product is equal to

$$\frac{dW}{dt} \propto -W + \left\langle yx^T \right\rangle_{p^\star} \Sigma^{-1} \tag{53}$$

Therefore, the exact natural gradient dynamics predict exponential convergence towards $W = \left\langle yx^T \right\rangle \Sigma^{-1}$, which is indeed the optimal solution of the linear regression problem, regardless of which specific generalized inverse is chosen for the Fisher matrix. Furthermore, exponential convergence holds only if weight matrices are updated following a natural gradient. Instead of the natural gradient in Eq. 47, we may update the weight matrices according to an arbitrary matrix C of size $P \times P$, namely

$$\text{Vec}\left(\frac{dW_1}{dt}, \ldots, \frac{dW_L}{dt}\right) = C \left\langle \frac{\partial \ell}{\partial \text{Vec}\left(W_1, \ldots, W_L\right)} \right\rangle_{p^\star} \tag{54}$$

Then, using Eqs.(48,51), the update of the total weight matrix product is given by

$$\text{Vec}\left(\frac{dW}{dt}\right) = \left(A^T C A\right) \text{Vec}\left\langle \frac{\partial \ell}{\partial x_L} x^T \right\rangle_{p^\star} \tag{55}$$

In order to achieve exponential dynamics for any output y (thus for any matrix $\left\langle yx^T \right\rangle$), we must have $A^T C A = B^{-1}$. There are infinitely possible matrices C satisfying this expression, and they can be written as $C = {A^l}^T B^{-1} A^l$, for all possible left inverses A^l, but this is indeed a generalized inverse of the Fisher matrix, Eq. 46.

8.5 A simple natural gradient for square matrices

We would like to obtain a simple expression for the natural gradient across the single layer matrices $W_1, \ldots, W_L$, using Eq. 49. We note that, if all weight matrices are square and invertible, one possible left inverse is given by (cf Eq. 43)

$$A^l = \frac{1}{L}\left(\left({W_1^b}^{-1} \otimes {W_1^a}^{T-1}\right), \ldots, \left({W_L^b}^{-1} \otimes {W_L^a}^{T-1}\right)\right) \tag{56}$$

We show that this expression implies exactly the update rule studied in the main text (Eq.10 and Eq.16 of the main text). Substituting Eqs.(32,44,56) into Eq. 49, we have

$$\text{Vec}\left(\frac{dW_i}{dt}\right) = \frac{1}{L} \text{Vec}\left({W_i^a}^{-1} \left\langle \left(y - x_L\right) x^T \right\rangle_{p^\star} \Sigma^{-1} {W_i^b}^{-1}\right) \tag{57}$$

This expression is equal to Eq. 16 of the main text, thus demonstrating that indeed represents an instance of the natural gradient.

As mentioned in the main text, this exact expression is proportional to the block-diagonal approximation of the Fisher matrix, even though the Fisher matrix is not block-diagonal. In order to see this, we invert the diagonal blocks of Eq. 42 and multiply by the gradient Eqs.(35,32)

$$\left(F_{ii}\right)^{-1} \left\langle \frac{\partial \ell}{\partial \text{Vec}\left(W_i\right)} \right\rangle_{p^\star} = \text{Vec}\left({W_i^a}^{-1} \left\langle \left(y - x_L\right) x^T \right\rangle_{p^\star} \Sigma^{-1} {W_i^b}^{-1}\right) \tag{58}$$

which is equal to Eq. 57, besides a factor L^{-1}. In order to understand this puzzling observation, we look at the (i,j)-block of the generalized inverse Fisher matrix, using Eqs.(46), and substituting the left inverse of Eq. 56, that gives

$$(F^g)_{ij} = \frac{1}{L^2}\left(W_i^{b\,T^{-1}}\Sigma^{-1}W_j^{b^{-1}}\right)\otimes\left(W_i^{a^{-1}}\tilde{\Sigma}W_j^{a\,T^{-1}}\right) \tag{59}$$

Comparing this expression with the block-wise Fisher matrix, Eq. 42, we find that the (i,j)-block of the inverse Fisher is equal to the inverse of the (transposed) (j,i)-block of the Fisher matrix, besides a factor L^{-2}. Therefore, the Fisher matrix can be inverted by inverting single blocks individually. Furthermore, each block of the inverse contributes the same amount to the natural gradient, as shown by multiplying the last expression by the gradient of layer j (Eqs. 35, 32), namely

$$(F^g)_{ij}\left\langle\frac{\partial\ell}{\partial \mathrm{Vec}\,(W_j)}\right\rangle = \frac{1}{L^2}\mathrm{Vec}\left(W_i^{a^{-1}}\left\langle(y-x_L)\,x^T\right\rangle_{p^\star}\Sigma^{-1}W_i^{b^{-1}}\right) \tag{60}$$

This expression does not depend on j, therefore each block of the inverse Fisher, across columns, contributes equally to the natural gradient.

8.6 A simple update for rectangular matrices

In this section we show that, even if the weight matrices are not square and not invertible, Eq. 10 of the main text approximately results in exponential decay towards the minimum loss, provided that the Moore-Penrose pseudo-inverse is used when inverting covariances (Eqs. 14,15 in the main text).

In order to simplify the notation, we define the following matrices E_i, X_i

$$\left\langle e_i e_i^T\right\rangle_{p^\star} = E_i E_i^T \qquad\qquad E_i = (W_L\cdots W_{i+1})^T\,\tilde{\Sigma}^{-1/2} \tag{61}$$

$$\left\langle x_{i-1}x_{i-1}^T\right\rangle_{p^\star} = X_i X_i^T \qquad\qquad X_i = (W_{i-1}\cdots W_1)\,\Sigma^{1/2} \tag{62}$$

Using the properties of the pseudo-inverse of a product, we can compute explicitly the pseudo-inverse (denoted by the superscript $+$) of both covariance matrices, equal to

$$\left\langle e_i e_i^T\right\rangle_{p^\star}^{+} = E_i^{+T}E_i^{+} \qquad\qquad \left\langle x_{i-1}x_{i-1}^T\right\rangle_{p^\star}^{+} = X_i^{+T}X_i^{+} \tag{63}$$

Using the definitions of E_i and X_i, we note that the gradient Eq. 13 of the main text can be rewritten as

$$\frac{\partial\ell}{\partial W_i} = E_i\,\tilde{\Sigma}^{-1/2}\,(y-x_L)\,x^T\Sigma^{-1/2}\;X_i^T \tag{64}$$

We calculate the natural gradient update, Eq. 10 of the main text, with the pseudo-inverse in place of the inverse. Using Eqs.(63,64), that is equal to

$$\frac{dW_i}{dt}\propto\frac{1}{L}E_i^{+T}E_i^{+}E_i\,\tilde{\Sigma}^{-1/2}\left\langle(y-x_L)\,x^T\right\rangle_{p^\star}\Sigma^{-1/2}\;X_i^TX_i^{+T}X_i^{+} \tag{65}$$

Using the properties of the pseudo-inverse, we have that $E_i^{+T}E_i^{+}E_i = E_i^{+T}$ and $X_i^TX_i^{+T}X_i^{+} = X_i^{+}$. Furthermore, we use $x_L = Wx$, where we rewrite $W = \tilde{\Sigma}^{1/2}E_i^TW_iX_i\Sigma^{-1/2}$. Then the previous expression is rewritten as

$$\frac{dW_i}{dt}\propto\frac{1}{L}E_i^{+T}\left(\tilde{\Sigma}^{-1/2}\left\langle yx^T\right\rangle_{p^\star}\Sigma^{-1/2}-E_i^TW_iX_i\right)X_i^{+} \tag{66}$$

We would like to obtain an update for the total weight matrix product, similar to Eq. 18. Substituting Eq. 66 into the product rule Eq. 50, and using again the definitions of E and X, the update for the total weight matrix product is equal to

$$\frac{dW}{dt}\propto\frac{1}{L}\sum_{i=1}^{L}\tilde{\Sigma}^{1/2}E_i^TE_i^{+T}\left(\tilde{\Sigma}^{-1/2}\left\langle yx^T\right\rangle_{p^\star}\Sigma^{-1/2}-E_i^TW_iX_i\right)X_i^{+}X_i\Sigma^{-1/2} \tag{67}$$

Using again the properties of the pseudo-inverse, $E_i^TE_i^{+T}E_i^T = E_i^T$, and $X_iX_i^{+}X_i = X_i$, we finally obtain

$$\frac{dW}{dt}\propto -W+\frac{1}{L}\sum_{i=1}^{L}P_i^a\left\langle yx^T\right\rangle_{p^\star}\Sigma^{-1}P_i^b \tag{68}$$

where P_i^a and P_i^b are projection matrices, defined by $P_i^a = \tilde{\Sigma}^{1/2} \left(E_i^+ E_i\right)^T \tilde{\Sigma}^{-1/2}$ and $P_i^b = \Sigma^{1/2} X_i^+ X_i \Sigma^{-1/2}$. Therefore, the total product matrix converges exponentially to the optimal solution (cf Eq. 18 in the main text). The projection operators P_i^a and P_i^b depend on the weight matrices; the optimal solution is projected into a subspace which depends on the specific form of the deep network (e.g., whether is contracting, expanding, or has a bottleneck). Note that this result was obtained assuming that covariances in Eq. 10 of the main text are inverted using the Moore-Penrose pseudoinverse, and may not hold when using a different kind of inverse.